Data Sanitization: Improving the Forensic Utility of Anomaly Detection Systems

Gabriela F. Cretu, Angelos Stavrou, Salvatore J. Stolfo and Angelos D. Keromytis
Department of Computer Science, Columbia University
{gcretu, angel, sal, angelos}@cs.columbia.edu

Abstract

Anomaly Detection (AD) sensors have become an invaluable tool for forensic analysis and intrusion detection. Unfortunately, the detection performance of all learning-based ADs depends heavily on the quality of the training data. In this paper, we extend the training phase of an AD to include a sanitization phase. This phase significantly improves the quality of unlabeled training data by making them as "attack-free" as possible in the absence of absolute ground truth. Our approach is agnostic to the underlying AD, boosting its performance based solely on training-data sanitization. Our approach is to generate multiple AD models for content-based AD sensors trained on small slices of the training data. These AD "micro-models" are used to test the training data, producing alerts for each training input. We employ voting techniques to determine which of these training items are likely attacks. Our preliminary results show that sanitization increases 0-day attack detection while in most cases reducing the false positive rate. We analyze the performance gains when we deploy sanitized versus unsanitized AD systems in combination with expensive host-based attack-detection systems. Finally, we show that our system incurs only an initial modest cost, which can be amortized over time during online operation.

1 Introduction

A network-based intrusion detector can be used as an online traffic/input-filtering subsystem, or as a forensic tool to identify likely data that created a fault in a system after the fact. As "signature-based" network intrusion detection systems (NIDS) appear to become obsolete in detecting 0-day malicious traffic [6], effective anomaly detection that models normal traffic well remains an open problem. Ideally, an anomaly detector should achieve 100% detection accuracy, *i.e.,* true attacks are all identified, with 0% false positives. However, the particular modeling algorithm one uses to compute a model of "normal" data can be in error due to several problems:

- The model may have over-fit the training data and thus have poorly generalized the representation of "normal data". Thus, any data that is not observed during training can be regarded as anomalous, even if it may not be an attack. This leads to an alert being generated that is a "false positive".
- The training data may contain attacks that would "poison" the learned model of "normal" data allowing such attack data to be considered normal data, *i.e.* generating false negatives. Such problems have been noted by others in the past [8].

Hence, the alerts generated by an anomaly detector may or may not be true attacks. Many of these alerts may be false positives, an oftentimes cited problem with anomaly detectors [2]. We posit that the goals typically set forth for AD systems (reducing or entirely eliminating all false positives) are not the correct performance metric. Rather, the goal should be to create a detector that accurately identifies true normal, attack-free data, that may be processed directly by the intended service or application. All other data that generates alerts by the anomaly detector should be regarded as "suspect data" that would need to be further tested by another component to validate the true attacks from the false positives.

Our proposed approach is to better sanitize the training data, thereby computing a more accurate anomaly detector model that identifies suspect data properly and produces fewer false alerts. We assume that one can determine whether a packet is a true attack by using a heavily instrumented host-based "shadow" server system akin to a honeypot. By diverting all suspect data to this oracle, we will be able to identify true attacks by way of detecting malicious or abnormal actions performed by the server when processing the suspicious data. However, such a heavily instrumented shadow server is assumed to be (and in practice *is*) substantially slower (usually orders of magnitude slower) than the native application to

be protected [1] (another approach would be to correlate the alert with input from another anomaly detector [3]).

To address this problem, we generalize the notion of training for an AD system. Instead of using a single AD system trained on a single large set of training data, we use multiple AD instances. These instances are trained on smaller, non-overlapping slices of the original traffic dataset. This process produces models that represent a very localized view of the training data. We call such models *micro-models*.

Armed with the micro-models, we are now in a position to assess the quality of our training data and to automatically detect and remove any attacks or abnormalities that should not be considered part of the "normal" model. *The intuition behind this approach is the following: given a training set that spans a sufficiently large time interval, an attack or an abnormality appears only in a relatively confined window of time. To identify the attack, we use the micro-models in a voting scheme.* By applying this method on a second set of training data we obtain the *sanitized model,* which is very likely to be attack-free and therefore providing a boost in performance when used as input during the testing phase of the chosen anomaly detector.

Note that we are not claiming a complete solution to the false positive problem; instead, we pose a different performance objective for anomaly detectors, *i.e.*, within realistic operational environments, the key objective is to optimize the security and performance throughput of the system under protection. Our goal then is to limit the amount of network traffic data that would be safely processed by the shadow server, and to identify the normal, attack-free data that can processed by the native service or application without instrumentation. The performance objective is thus to:

- *accurately identify a set of "suspect data" in the input stream that is shunted to an expensive shadow server, which will verify whether the data in question is a true attack or a false positive artifact of the anomaly detector;*
- *maintain service throughput by limiting the amount of data that will be subjected to such host-based tests, which would slow down response rates for only that portion of the input stream deemed abnormal.*

2 Sanitization Architecture

Cleaning temporally ordered training data is a crucial first step for training any learning-based anomaly detector. Supervised training (using labeled datasets) may be ideal, but it is generally infeasible in view of the amount of data that needs to be cleaned or validated. We conjecture that attacks would be a minority class of data[1], and thus having a large training set increases the probability that an individual datum is normal. With increasing amounts of training data, the probability of malcode presence also increases. This malcode data may poison the model and complicates the task of classifying normal data.

The corollary to our conjecture is that while the time an attack appears in a training set is unknown, the attack itself will manifest as a few packets that will not persist throughout the dataset. Common attack packets tend to cluster together over small periods, forming a sparse representation over time. For example, once a worm outbreak starts it will be concentrated in a relatively short period of time, and eventually the infected hosts will be either patched, rebooted or filtered, causing less appearance of that worm in the dataset [4]. Again, we stress the fact that these assumptions hold true over relatively long periods of time, necessitating the use of large training datasets to properly sanitize an AD model.

2.1 Micro-models

Based on our observations, if we partition a large training dataset into a number of smaller, time-delimited training sets, we may compute a minority set of partitions (micro-datasets) that contain attack vectors. Each of these time-based "epochs" is used to compute a 'micro-model' using any chosen anomaly detection algorithm. Hence, for time period (epoch) t_i we compute model M_i. Given our assumptions, a distinct attack vector that may appear during time period t_j will affect the model computed for that time period, hence M_j may be poisoned, having modeled the attack vector as normal data during epoch t_j, but model M_k computed for time period t_k, $k \neq i$ would not be poisoned.

To maximize our likelihood of finding a set of micro-models that are not poisoned by attack data, we need to identify the right level of time granularity. The training epochs can naturally vary over the entire set of traffic data captured. In our experimental work reported later, we analyzed packets traces captured over hundreds of hours and found that 3 to 5 hours of packet capture was sufficient to generate well-behaved micro-models (they do not generate large numbers of false positives).

2.2 Sanitized and Malicious Models

In the second phase, we compute a new AD model by using previously built micro-models to sanitize training data. We start with a new dataset, which is tested by

[1] While the total attack volume in any given trace may be high, the frequency of specific attacks is generally low relative to the legitimate inputs. This may not be the case in certain circumstances, *e.g.*, during a DDoS attack or during the propagation phase of a fast worm such as Slammer. It may be possible to identify such non-ideal conditions for AD training by analyzing the entropy of a particular dataset (too high or too low may indicate exceptional circumstances). We leave this analysis for future work.

instances of the proposed AD against the micro-models M_i. Each of these tests results in a labeled data set L_i, with each packet labeled as *normal* or *abnormal*. We note that these labels are not yet generalized, being representative of the micro-model that performed each test. The labeled data sets are then processed through a voting scheme, which assigns the final label to each packet. Consider the case where a micro-model M_a includes the information of an attack; when used for testing, it will probably label a packet containing that particular attack vector as *normal*. However, given our conjectures, only a minority of the micro-models will include the same attack vector as M_a. Thus, we use voting strategies, where packets are labeled as normal if **a weighted majority** of all AD instances will agree on this. As a result, we create two disjoint datasets: one that contains only majority-voted normal packets from which the "sanitized model" is built, and the rest, from which a "model of malicious attack data" is computed.

For the voting algorithms we chose two simple strategies presented below. Investigating and evaluating other voting strategies is an interesting item for future work.

- **naive voting**, which assigns to each AD_i the same weight in the voting process. A packet is deemed to be normal if:

$$\frac{1}{N}\sum_{i=1}^{N} a_i \leq V$$

where $a_i = 0$ if AD_i finds the packet normal; otherwise $a_i = 1$.

- **weighted voting**, which assigns to each micro-model M_i a weight w_i equal to the number of packets used to train it. A packet is considered normal if:

$$\frac{1}{\sum_{j=1}^{N} w_j}\sum_{i=1}^{N} w_i \cdot a_i \leq V$$

The threshold *V* is the number of votes required for a packet to be considered abnormal. We have to have that $1 - V > N_p$, where N_p is the maximum number of models poisoned by any specific attack vector.

After the training phase, we can use the sanitized model with the anomaly detector for online testing. Recall, our approach is agnostic to the anomaly detection algorithm, generating only a sanitized model that is then used for testing. As a result, our scheme can be applied to a wide range of ADs, as illustrated in the following section.

3 Evaluation

In this section, we experimentally quantify the increase in the detection accuracy of out-of-the-box content-based anomaly detection systems when we apply training data sanitization. The goal of our system is to detect all true attacks and at the same time maintain or even reduce the total number of generated alerts. We evaluate our approach using two different scenarios. In the first scenario, we measure the performance of the AD sensor with and without sanitization. Additionally, we consider the case where we use the AD as a packet classifier for incoming network traffic: we test each packet and divert all packets that generate an alert to a back-end shadow server. Both the feasibility and scalability of this scenario depend mainly on the amount of alerts generated by the AD sensor, since all "suspect-data" are delayed significantly by the shadow server and such data come from both real attacks and false alerts.

For our experiments, we use two content-based anomaly detectors Anagram [10] and Payl [9]. Both detectors, like most anomaly detection sensors, have a training and testing phase. For training phase, we supply the AD sensor with "normal" data used to build a normality model. This phase is crucial to the sensor's future detection performance: the detector could compute a faulty model if provided with training data that happens to include malicious or attack packet content. This becomes more apparent if we notice that anomaly detection sensors operate on the principle that attacks, and particularly zero day attacks, manifest as packet datagrams never before seen during training. Although dependent on a clean initial model, AD sensors have quite different learning algorithms to determine whether they have seen a particular datum before or not. We do not describe the details of the algorithms used during the testing phase, as they are not germane to the topic of this paper; the interested reader is referred to the citations above.

Our experimental corpus consists of 500 hours of real network traffic, which translates into approximately four million content packets. We split these data into three separate sets: two used for training and one used for testing. We use the first 300 hours of traffic to build the micro-models and the next 100 hours to generate the sanitized model. The remaining 100 hours of data, consisting of approximately $775,000$ packets, were used for testing. In addition, to validate our results, we used the last 100 hours to generate the sanitized model while testing on the other 100-hour dataset.

3.1 Experimental Results

In our initial experiment, we measured the detection performance for both Anagram and Payl when used as stand-alone online anomaly detectors. Then, we repeated the same experiments using the same setup and network traces but including the sanitization phase. Table 1 presents our findings, which clearly show that by using a sanitized training dataset, we boost the detection capabilities of both AD sensors. Observe that we maximize the detection of the real alerts while generating very low false positives rates. It is important to notice that without

sanitization, the normal models used by Anagram were poisoned with attacks and thus unable to detect new attack instances appearing in the testing data. Therefore, making the AD sensor more sensitive, *e.g.* changing its internal detection threshold, would only increase the false alerts without increasing the detection rate. In this experiment, the traffic contains instances of phpBB forum attacks (mirela, cbac, nikon, criman).

Table 1: AD sensors comparison

Sensor	FP rate	TP rate
Anagram	0.00074%	0%
Anagram with Snort	0.00214%	29.29%
Anagram with Sanitization	**0.00089%**	**100%**
Payl	0.00849%	0.0%
Payl with Sanitization	**0.00373%**	**29.29%**

We reiterate the fact that our training dataset was indeed poisoned with attack vectors and thus the worm packets were also included in the normal model, which resulted in detection rate of 0%. When using previously known malcode information (using Snort signatures represented in a "malicious model"), Anagram was able to detect a portion of the worm packets. Of course, this detection model is limited because it requires that a new 0-day worm will not be sufficiently different from previous worms that appear in the traces. To make matters worse, such a detector would fail to detect even old threats that do not have a Snort signature. On the other hand, if we enhance Anagram's training phase to include sanitization, we do not have to rely on any other signature or content-based sensor to detect malware.

Furthermore, the detection ability of a sensor is inherently dependent on the actual algorithm used to compute the distance of a new worm from the normal model. For example, although Payl is effective at capturing attacks that display abnormal byte distributions, it is prone to miss well-crafted attacks that resemble the byte distribution of the target site [10]. Our traces contain such attacks, which is the reason why, when we use the sanitized version of Payl, we can only get a 30% worm detection rate as opposed to 100%. The sanitization phase is a necessary requirement in detecting malcode but not a sufficient one: the actual algorithm used by the sensor is also very important in determining the overall detection capabilities of the sensor.

Overall, our experiments show that the AD signal-to-noise ratio (*i.e.*, true positives over false positives) can be significantly improved even in extreme conditions, when intrinsic limitations of the anomaly detector prevent us from obtaining a 100% attack detection.

To stress our system and to validate its operation, we also performed experiments using traffic in which we artificially injected worms such as CodeRed, CodeRed II, WebDAV, and a worm that exploits the nsiislog.dll buffer overflow vulnerability (MS03-022). All instances of the injected malcode were recognized by the anomaly detectors when trained with sanitized data. That re-enforced our initial observations about the sanitization phase: we can both increase the probability of detecting a zero-day attack and of previously seen malcode.

3.2 Performance Evaluation

Another aspect of an anomaly detection system that we would like to analyze is its impact on the average time that it takes to process a request. In addition, we measure the overall computational requirements of a detection system consisting of an AD sensor and a host-based sensor (shadow server). The AD sensor acts as a packet classifier diverting all packets that generate alerts to the host-based sensor while allowing the rest of the packets to reach the native service. Our goal is to create a system that does not incur prohibitive increase in the average request latency and at the same time can scale to millions of service requests. Therefore, we would like the AD to shunt only a small fraction of the total traffic to the expensive shadow servers.

In our experimental setup, we used two well-known instrumentation frameworks: STEM [5] and DYBOC [1]. STEM exhibits a 4400% overhead when an application such as Apache is completely instrumented to detect attacks. On the other hand, DYBOC has a lighter instrumentation, providing a faster response, but still imposes at least a 20% overhead on the server performance. We conducted our tests on a PC with a 2GHz AMD Opteron processor with 8GB of RAM, running Linux.

To calculate the overall overhead, we used the same method used in [10]. We define the latency of such an architecture as following: $l' = (l*(1-fp))+(l*O_s*fp)$, where l is the standard (measured) latency of a protected service, O_s is the shadow server overhead, and fp is the AD false positive rate.

To quantify the performance gain from using the sanitization phase, we compare the average latency of the system when using Payl and Anagram with sanitized and non-sanitized training data. From Table 2, we see that for both sensors the alert rate will decrease after sanitizing the training data, and fewer number of packets will have to be processed by the shadow server.

Table 2: Latency for different anomaly detectors

Sensor	STEM	DYBOC
No-sensor	$44 * l$	$1.2 * l$
Anagram with Snort	$1.092 * l$	$1.00042 * l$
Anagram with sanitization	$\mathbf{1.042 * l}$	$\mathbf{1.00017 * l}$
Payl	$1.365 * l$	$1.00169 * l$
Payl with sanitization	$\mathbf{1.160 * l}$	$\mathbf{1.00074 * l}$

Finally, we estimate the gain in the system performance when using the sanitization process. We define the overall processing impact ratio as $(N_{san}/N_{non\_san}) * 100$, where N_{san} is the number of alerts sent to the shadow server when the sanitized model is used and N_{non_san} is the number of alerts for the non-sanitized version. For Anagram, the gain in performance is 52.27%, and for Payl 44.44%. This shows that sanitization reduces by half the computational cost to process the same amount of data, leading to a two-fold increase in scalability of the overall architecture independent of the AD system used.

4 Collaborative Sanitization

Our scheme continuously revises AD models based on prior history and the voting scheme. Based on this, it is conceivable that a determined adversary can launch an extensive and long lasting training and targeted attack, which would poison all the micro-models. Such a scenario would run counter to our conjecture from Section 2. To counter such attacks, we propose as future work to extend our sanitization mechanism to allow the sharing of malicious models [7] generated by collaborating remote sites, and to sanitize the local training data to a greater extent. These models, which can be privacy-preserving, capture characteristics of malicious behavior (rather than normal behavior, which is the default AD operation). This approach may not apply to polymorphic attacks, since each propagation attempt will display a distinct attack vector that may be captured in different malicious models. We conjecture, however, that a polymorphic attack "targeting a single site" can still be captured by the local sanitization scheme presented in this paper. However, it remains to be seen (through additional testing) how well our scheme (with or without the collaborative sanitization extensions) can cope with polymorphism or long-term training attacks.

5 Conclusions

We introduce a novel sanitization method that boosts the performance of out-of-the-box anomaly detection sensors, elevating them to a first-rate forensics and alert analysis tool. Our approach is simple and general, and can be applied to a wide range of unmodified AD sensors without incurring significant additional computational cost other than the initial testing phase. Preliminary experimental results indicate that our system can serve as an efficient and accurate online packet classifier. The alerts generated by the "sanitized" AD model are a small fraction of the total traffic, and produce 1/2 as many alerts as the original unsanitized AD model. Furthermore, the AD system is capable of detecting more threats both online and after an actual attack, since the AD training data are attack-free. We have also identified several areas for future work and possible extensions, as well as further directions for evaluation and improvement.

References

[1] K. G. Anagnostakis, S. Sidiroglou, P. Akritidis, K. Xinidis, E. Markatos, and A. D. Keromytis. Detecting Targeted Attacks Using Shadow Honeypots. In *Proceedings of the* 14^{th} *USENIX Security Symposium*, August 2005.

[2] H. Debar, M. Dacier, and A. Wespi. Towards a taxonomy of intrusion-detection systems. In *The International Journal of Computer and Telecommunications Networking*, volume 31, pages 805–822, 1999.

[3] O. Kreidl and T. Frazier. Feedback control applied to survivability: a host-based autonomic defense system. In *IEEE Transactions on Reliability*, volume 53, pages 148–166, 2004.

[4] D. Moore and C. Shannon. The Spread of the Code Red Worm (CRv2). `http://www.caida.org/analysis/security/code-red/coderedv2_analysis.xml`.

[5] S. Sidiroglou, M. E. Locasto, S. W. Boyd, and A. D. Keromytis. Building a Reactive Immune System for Software Services. In *Proceedings of the USENIX Technical Conference*, June 2005.

[6] Y. Song, M. E. Locasto, A. Stavrou, A. D. Keromytis, and S. J. Stolfo. On the infeasibility of Modeling Polymorphic Shellcode for Signature Detection. In *Columbia University Computer Science Department Technical Report, CUCS 007-07*, 2007.

[7] S. Stolfo, W. Fan, W. Lee, A. Prodromidis, and P. Chan. Cost-based Modeling for Fraud and Intrusion Detection: Results from the JAM Project. In *Proceedings of the DARPA Information Survivability Conference and Exposition (DISCEX)*, 2000.

[8] K. M. Tan and R. A. Maxion. Why 6? Defining the Operational Limits of stide, an Anomaly-Based Intrusion Detector. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 188–201, May 2002.

[9] K. Wang, G. Cretu, and S. J. Stolfo. Anomalous Payload-based Worm Detection and Signature Generation. In *Proceedings of the Symposium on Recent Advances in Intrusion Detection (RAID)*, September 2005.

[10] K. Wang, J. J. Parekh, and S. J. Stolfo. Anagram: A Content Anomaly Detector Resistant to Mimicry Attack. In *Proceedings of the Symposium on Recent Advances in Intrusion Detection (RAID)*, September 2006.